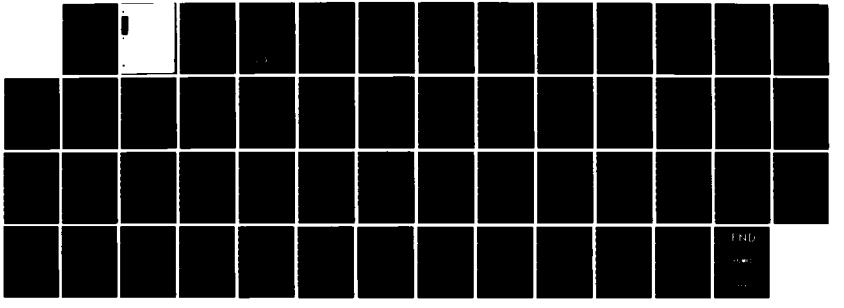
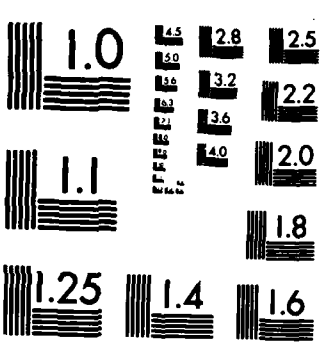
AD-A154 339 MC TRICOMP USER'S GUIDE(U) NAVAL SURFACE WEAPONS CENTER 1/1
DAHLGREN VA J A GAINES JUL 84 NSWC/TR-84-95
SBI-AD-F350 016

UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NSWC TR 84-95	2. GOVT ACCESSION NO. AD-A154339	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) MC TRICOMP USER'S GUIDE		5. TYPE OF REPORT & PERIOD COVERED Final 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Jack A. Gaines, Jr.		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Surface Weapons Center (Code K53) Dahlgren, VA 22448		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 36801
11. CONTROLLING OFFICE NAME AND ADDRESS Strategic Systems Program Office Washington, DC 20376		12. REPORT DATE July 1984 13. NUMBER OF PAGES 53
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution is unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

TRIDENT Higher Level Language	Compiler
THLL	program
MC68000	implementation
VAX 11/780	runtime system

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The TRIDENT Higher Level Language (THLL) is implemented on the VAX 11/780 based TRIDENT Software Generation System (SGS) for three target machines: the TRIDENT Basic Processor (BP), the Motorola MC68000, and the VAX 11/780. This document describes implementation features of THLL, which are unique for the MC68000, and the runtime support system for the MC68000.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

FOREWORD

This document was written in the Operational Support Branch (K53), Submarine Launched Ballistic Missile (SLBM) Software Development Division (K50), of the Strategic Systems Department (K) at the Naval Surface Weapons Center (NSWC), Dahlgren, Virginia.

The purpose of this document is to describe implementation features of the TRIDENT Higher Level Language (THLL), which are unique for the Motorola MC68000, and the runtime support system for THLL on the MC68000. It supplements the THLL Reference Manual for programs designed to run on the MC68000.

This report was extensively reviewed by personnel in the Operational Support Software Systems Group in K53, the Quality Assurance Branch (K52), and the Operational Systems Branch (K54).

This project was funded by the Strategic Systems Program Office, Washington, DC 20376, under task number 36801.

Questions, comments, and suggestions concerning the material presented in this document should be directed to the Commander, Naval Surface Weapons Center, ATTN: K53, Dahlgren, Virginia 22448.

Approved by:

THOMAS A. CLARE, Head
Strategic Systems Department

CONTENTS

CHAPTER 1

INTRODUCTION

MC TRICOMP is a compiler for the TRIDENT Higher Level Language (THLL) that runs on the VAX 11/780 and produces code for the Motorola MC68000. This document describes implementation features of THLL, which are unique for the MC68000, and the runtime support system for the MC68000.

Differences between the MC68000 compiler and other THLL compilers are due to differences in the hardware of the target machines and differences in the software runtime environment of the target machines. This document is not self-contained. It is to be used in combination with the THLL Reference Manual (Reference 1). Details about the MC68000 architecture can be found in the MC68000 16-Bit User's Manual (Reference 2).

MC TRICOMP allows the use of THLL as a high level programming language for the MC68000, a number of which will be included in the TRIDENT II Fire Control System. It supports the full THLL language. However, THLL Input/Output (I/O) and some predefined procedures are not supported in the MC68000 runtime library. MC TRICOMP is an interim compiler that will be phased out when the THLL II MC TRICOMP comes into existence.

The MC68000 architecture views data as 8-bit bytes, 16-bit words, and 32-bit longwords. The address of the data is the byte address of the first byte of the data. THLL views data in 32-bit units called THLL words (which are MC68000 longwords). Throughout this User's Guide, the term "word" refers to a 32-bit THLL word and is not to be confused with the MC68000 16-bit word. Also, a THLL doubleword (or simply doubleword) is a contiguous pair of THLL words.

CHAPTER 2

BASIC ELEMENTS OF THLL

2.1 THLL CHARACTER SET

The THLL character set consists of the following ASCII subset:

Uppercase letters A-Z

Numerals 0-9

The following special characters:

Character	Name	Character	Name
	Space	.	Period
!	Exclamation point	/	Slash
"	Quotation mark	:	Colon
#	Number sign	;	Semicolon
$	Dollar sign	<>	Angle brackets
%	Percent sign	=	Equal sign
&	Ampersand	?	Question mark
'	Apostrophe	@	At sign
()	Parentheses	[]	Square brackets
*	Asterisk	\	Backslash
+	Plus sign	^	Circumflex
,	Comma	_	Underline
-	Minus sign		

Not included in the THLL character set are the lowercase letters (codes X'60' - X'7E') and the nonprintable characters (codes X'00' - X'1F' and X'7E'). When these characters are used in a THLL source file they are treated as follows:

A. Lowercase letters are converted to uppercase letters (codes X'40' - X'5E') except when they appear in comments or remarks.

B. Nonprintable characters are converted to question marks (?).

The listing file produced by MC TRICOMP reflects these character conversions.

A table of ASCII characters is included in Appendix A.

2.2 OPERATORS

The value of LOC X is the actual machine address of the THLL word or doubleword containing X. The machine address is a 32-bit quantity of type POINTER.

The value of LOCA X is also the actual machine address of the THLL word or doubleword containing X. In this case, the type of the value is INTEGER. The bit patterns of LOC X and LOCA X are identical.

2.3 DELIMITERS

All of the THLL delimiters are implemented in MC TRICOMP, but some have little or no meaning in the environment of the MC68000. Examples are: EXEC, INTERRUPT, LINK.

2.4 IDENTIFIERS

The first 31 characters of identifiers are significant. The first character of the identifier must be a letter. The remaining characters can be letters, numerals, or special characters. The last character of an identifier cannot be a special character.

MC TRICOMP accepts dot (.), underscore (_), and question mark (?) as special characters within identifiers. A dot that occurs in an external symbol is mapped into a question mark. In this case, uniqueness is not guaranteed between identifiers that contain either of these characters. Special care should be taken for identifiers using either the dot or the question mark. For example, X.Y and X?Y are not unique if both identifiers are external symbols.

Care must be taken not to choose global or external names that have other meanings to the MC68000 assembler. Examples are: SP, PC, D0 thru D7, and A0 thru A7.

2.5 CONSTANTS

For a pictorial representation of the layout of the following constants, see Section 3.3.

2.5.1 Half, Integer, and Double Constants

In MC TRICOMP, any decimal integer that exceeds 31 bits of significance is considered to be a double value. Any decimal integer that does not exceed 16 bits of significance is considered to be a half value. Binary, octal, and hexadecimal integers are considered to be double when they exceed 32 bits of significance. They are considered to be half when they do not exceed 16 bits of significance.

A half constant is kept in a THLL halfword (16-bit MC68000 word). An integer constant is kept in one THLL word (32-bit MC68000 longword). A double constant is represented by a THLL doubleword (two 32-bit MC68000 longwords).

It should be noted that the following two statements are not equivalent in MC TRICOMP (INT is type INTEGER):

```
INT = X'0FFFF' ;    /* INT IS SET TO X'0000FFFF' */
INT = X'FFFF' ;     /* INT IS SET TO X'FFFFFFFF' */
```

The lack of equivalence is due to the sign extension of a half constant when assigned to an integer variable. This sign extension can also have an effect when comparing integer and half values.

2.5.2 Real Constants

MC TRICOMP uses a 48-bit item stored in a 64-bit space to represent a THLL real number. The 48 bits consist of a 16-bit two's complement exponent followed by a 32-bit two's complement mantissa (i.e., a truncated BP format). The range for real constants on the MC68000 is approximately .353E-9864 thru .708E+9864. The precision for real constants on the MC68000 is approximately 9 decimal digits.

2.5.3 Pointer Constants

The only legal pointer constant is 0. Pointer expressions built up using THLL operators are valid at preset time and runtime. A pointer contains the byte address of the designated item. Since the memory addressing capability of the MC68000 is very large, pointer components on the MC should be 32 bits wide.

2.5.4 Boolean Constants

TRUE is defined to be a 32-bit integer that has all 32 bits set; however, any non-zero value tests as TRUE. FALSE is represented as X'00000000'.

2.5.5 Strings

A string is represented by a header word followed by a block of words holding the characters of the string. The characters are ASCII as defined for the MC68000. Each character occupies one 8-bit byte. The characters are packed from left to right within a THLL word.

CHAPTER 3

DATA DECLARATIONS

3.1 TYPES

There are six types for classifying constants, variables, and procedures in THLL. A type may be thought of as a class of values. In general, the user should only have to know about the abstract properties of a type, the rules for the use of data types by operators and procedures, and the rules for type conversion. For portability reasons, the user should make a conscious effort to not take advantage of a particular implementation of types. In some cases, however, it may be necessary to know about the internal representation of the various types of values in memory.

Constants are assigned types as specified in Sections 2.5.1 through 2.5.5. Symbol types are specified in the corresponding declarations. Valueless items, such as statements, are assigned no type (N).

3.2 BIT NUMBER CONVENTIONS

THLL bit numbers run from 0 for the leftmost (most significant) bit to 31 for the rightmost (least significant) bit of a THLL word. For a doubleword the bit numbers run from 0 to 63.

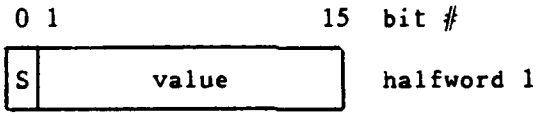
This convention is used for defining the meaning of a component "start bit" and "number of bits." Functions such as TEST.BIT, CLR.BIT, SET.BIT, TGL.BIT, and FIND.BIT use the THLL bit number as an input parameter, and FIND.BIT can also output a THLL bit number.

THLL bit numbers are opposite of the MC68000 view of bit numbers. The MC68000 bit numbering convention may need to be used when interfacing with other languages.

3.3 STORAGE FORMAT

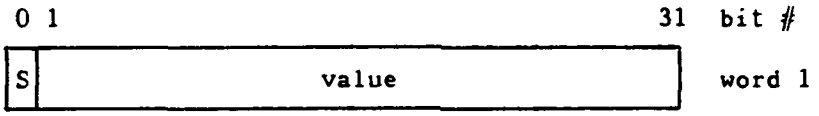
The types HALF (H), INTEGER (I), DOUBLE (D), and REAL (R) have a fixed format as to number of bits, location of sign bit, and number of words required for storage.

Type HALF requires one halfword, 16 bits.

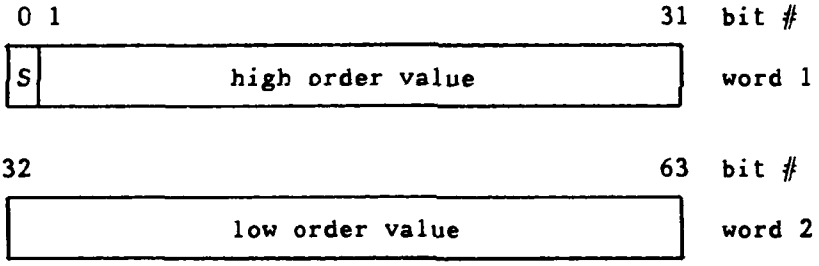


S = sign bit

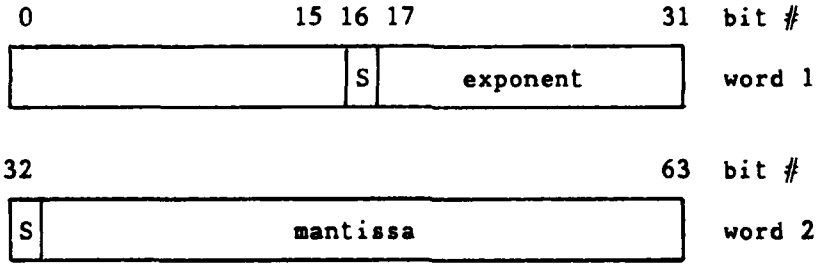
Type INTEGER requires one word, 32 bits.



Type DOUBLE requires two words, 64 bits.



Type REAL requires two words, 64 bits, separated into exponent and mantissa fields. The exponent and the mantissa are both two's complement values. The most significant 16 bits of the first word are not used.



The allocation of storage within a common is the same as the allocation of storage outside the common.

3.5 COMPONENTS

Even though the addressing on the MC68000 is in units of bytes, THLL components are based on the concept of THLL words. The offset part of the component declaration is in units of THLL words. When an integer expression is added to the pointer in a component reference, it is also considered to be in units of THLL words. MC TRICOMP automatically multiplies these values by 4 to access the correct byte address.

The potential address space of the MC68000 is quite large. Therefore, POINTER components on the MC68000 should be 32 bits wide.

REAL and DOUBLE components reference THLL doublewords. However, they do not need to be aligned at doubleword addresses. The offset in the component declaration is in units of THLL words. Thus, a DOUBLE or REAL component occupies two offset values.

3.5.1 Indexed Components

Full word INTEGER and POINTER components index in units of THLL words, and DOUBLE and REAL components index in units of THLL doublewords.

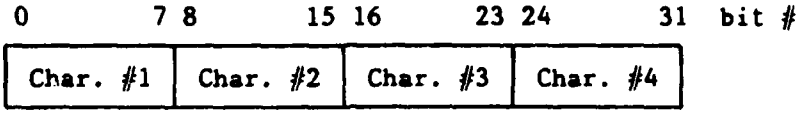
Partial word indexed components run with increasing index from left to right. For portability reasons, characters in ALPHA or DOUBLE variables should not be accessed with INTEGER indexed components. Instead, ALPHA components should be used.

3.5.2 Alpha Components

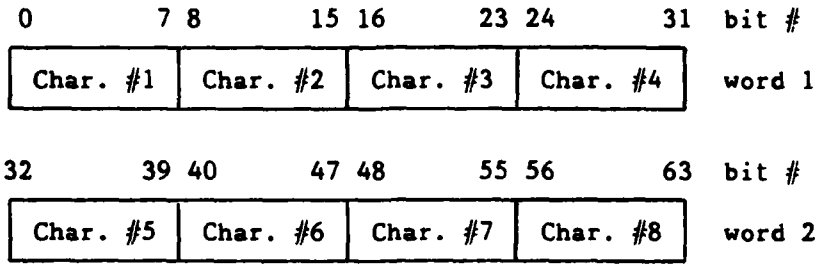
The ALPHA component provides a machine independent method of accessing character data. In effect, it is an indexed INTEGER component that indexes in the same direction as character data on the target computer. The type of the value of an alpha component is INTEGER, not ALPHA!

An ALPHA component variable must always be used as an indexed component variable. The OFFSET information in the component declaration is in units of THLL words. A particular character within the THLL word can be accessed by using an appropriate character number as in the example below.

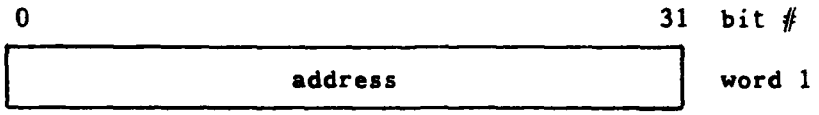
Type ALPHA is stored as four characters per word plus one header word. Thus, the number of words required for a string of n characters is ((n+3)/4) + 1. However, the programmer does not need to allow for the header when specifying string length.



Eight characters of a string can also be stored in a variable of type DOUBLE.



Type POINTER requires one word, 32 bits.



3.4 ALLOCATION OF STORAGE

Storage is allocated in the order in which data are declared. All THLL data structures are allocated at THLL halfword, word, or doubleword addresses. A THLL halfword starts at a byte address which is a multiple of 2. A THLL word starts at a byte address which is a multiple of 4. A THLL doubleword starts at a byte address which is a multiple of 8. HALF simple variables are allocated at halfword addresses. All data structures of type DOUBLE or REAL and all stacks and commons are allocated at doubleword addresses. All other data structures are allocated at word addresses. All constants are allocated at word addresses.

If in allocating a doubleword address, a "hole" arises, the "hole" remains. No attempt is made to fill the hole.

```
ALPHA COMPONENT CHR ;
OFFSET 0 FOR CHR ;     /* OPTIONAL - 0 IF NOT SPECIFIED */
                 or
ALPHA COMPONENT CHR(OFFSET 0) ;

usage:  CHR(P,I)       /* P - POINTER TO FIRST WORD THAT
                             CONTAINS CHARACTER DATA */
                       /* I - CHARACTER NUMBER */
```

Note that the following two declarations are equivalent in MC TRICOMP:

```
ALPHA COMPONENT CHAR (OFFSET 0) ;
INTEGER COMPONENT BYTE (FIELD(0,8),OFFSET 0) ;
```

The two component variables CHAR(P,I) and BYTE(P,I) always have the same value on the MC68000. However, the second method is not recommended.

3.6 PRESETS

Compile time expressions evaluated by MC TRICOMP are the same as runtime expression evaluation on the MC68000. Presets using indexed components work the same as runtime indexed components.

The preset functions LINKWORD(LOC P,N) and INITWORD(LOC P,N) each generate a THLL word of type INTEGER containing the address of procedure P. The N parameter of these functions is accepted by MC TRICOMP for compatibility with other THLL compilers, but it is not used in the preset.

CHAPTER 4

EXPRESSIONS AND STATEMENTS

4.1 EXPRESSION TYPES

Pointers contain addresses in units of bytes. THLL considers memory to be in units of 32-bit THLL words. Offsets from a pointer value are in units of THLL words. Whenever an integer expression is added to or subtracted from a pointer, MC TRICOMP automatically multiplies the integer value by four. When the difference of two pointer values is desired, MC TRICOMP divides the integer result by four.

An expression of type HALF is represented by a 16-bit THLL halfword. This includes constants in the appropriate range, computed values, and all items declared to be type HALF.

4.2 PROCEDURES

A THLL procedure can be called from another THLL procedure, an assembly program, or the operating system. A THLL procedure can call itself, other THLL procedures, and assembly programs.

THLL allows recursion, THLL procedures running on the MC68000 can be executed in a recursive manner. Variables sensitive to each invocation of a procedure must be shared (non-OWN).

Appendix B contains details about the runtime environment on the MC68000. Appendix C shows a method for preparing programs for the MC68000.

4.2.1 LINK and EXEC Procedures

LINK and EXEC procedures are assumed to be GLOBAL. It is not considered an error if they are also declared to be GLOBAL.

Any global procedure can be used as the entry point for executing the object program. The MAIN directive is used to specify the execution entry point.

4.2.2 Optional Arguments

The THLL functions, ARGTYPE and ARGSYNCL, use an argument descriptor list. In order for this list to be formed, OPTARG must be specified in the parameter list of the procedure that uses these functions. Likewise, if this procedure is called from another compile unit, the external declaration for this procedure in that compile unit must also contain the OPTARG specification.

ARGSYNCL is used to determine the syntactic class of the I'th argument of a call to a procedure using optional arguments. More information about ARGSYNCL may be found in the THLL Reference Manual (Reference 1). For the MC68000, the meaning of the integer value returned by ARGSYNCL is:

Argument Class	Value Returned
one-dimensional array	1
two-dimensional array	2
three-dimensional array	3
simple variable	4
stack	5
procedure	6
device	7
format	8

CHAPTER 5

THLL I/O ON THE MC68000

The MC68000 does not have any THLL Input/Output (I/O) support. I/O statements in a THLL compile unit compile correctly. However, no runtime support is provided for the calls generated by these statements. THLL I/O is not required on the MC68000's included in the TRIDENT II Fire Control System.

CHAPTER 6

STANDARD PROCEDURES

All standard procedures described in the THLL Reference Manual (Reference 1) are available on the MC68000.

Most of the standard functions and a number of operators such as ** are implemented as routines in the THLL runtime library, MCSYSLIB. Appendix C shows how to access this library.

The following sections describe MC68000 machine dependencies for the standard procedures.

6.1 FIXH, FIXI, AND FIXD FUNCTIONS

The value returned is the greatest integer less than or equal to the argument. Thus FIXI(-4.5) = -5 and FIXI(4.5) = 4. This is also the case when the compiler automatically generates a REAL to INTEGER type conversion.

6.2 MATHEMATICAL FUNCTIONS

The mathematical routines such as the trigonometric functions, SQRT, LN, EXP, etc. are supported either by firmware or by the runtime library, MCSYSLIB. The response by these routines to illegal parameter values depends on the MC68000 floating point firmware package. They do not necessarily respond the same as described in the THLL Reference Manual.

6.3 SHIFT FUNCTIONS

A HALF operand is shifted as a 16-bit quantity.

6.4 SWA FUNCTION

The SWA function is used only as a means of changing the type of a number from type INTEGER to type POINTER.

CHAPTER 7

UTILITY PROCEDURES

Each utility procedure that is used must be declared external. The form for that external declaration appears with each procedure. The effect, restrictions, and applications are briefly defined. These procedures can be found in the THLL runtime library, MCSYSLIB. Appendix C shows how to access this library.

7.1 EPRINT

EPRINT prints a line on OUTPUT. The form of the procedure call is:

```
EPRINT(MSG)
```

where

MSG - the ALPHA string to be printed or a pointer to the ALPHA.

The MSG is printed on OUTPUT in the following format:

```
**** ERROR AT LABEL <string>
```

The location of OUTPUT depends whether EPRINT executes on the Microtec MC68000 Simulator (Reference 4) or on an actual MC68000 with the MACSBUG operating system. On the Simulator, OUTPUT is the file SYS$OUTPUT. On an actual MC68000, OUTPUT is a printer connected to the processor output port.

The external declaration is:

```
EXTERNAL PROCEDURE EPRINT(ALPHA) ;
```

7.2 TERM

TERM prints a line on OUTPUT. The form of the procedure call is:

```
TERM(MSG)
```

where

MSG - the ALPHA string to be printed or a pointer to the ALPHA.

The location of OUTPUT depends whether EPRINT executes on the Microtec MC68000 Simulator (Reference 4) or on an actual MC68000 with the MACSBUG operating system. On the Simulator, OUTPUT is the file SYS$OUTPUT. On an actual MC68000, OUTPUT is a printer connected to the processor output port.

The external declaration is:

```
EXTERNAL PROCEDURE TERM(ALPHA) ;
```

7.3 TRUNC

TRUNC is an integer procedure that can be used to obtain the singleword representation of the integer portion of a real expression. For example, TRUNC(2.5) is 2, and TRUNC(-2.5) is -2. This is in contrast to the FIX functions described in Section 6.1.

The form of the procedure call is:

```
TRUNC(R)
```

where

R - any real valued expression

It should be noted that no attempt is made to determine whether the value of the integer obtained can fit into a singleword.

The external declaration is:

```
EXTERNAL INTEGER PROCEDURE TRUNC(REAL VALUE) ;
```

7.4 TRUNCD

TRUNCD is a double procedure that can be used to obtain the doubleword representation of the integer portion of a real expression.

The form of the procedure call is:

```
TRUNCD(R)
```

where

R - any real valued expression

The external declaration is:

```
EXTERNAL DOUBLE PROCEDURE TRUNCD(REAL VALUE) ;
```

CHAPTER 8

REFERENCES

1. Hartmut G. Huber, THLL Reference Manual, NSWC TR 84-101, Jul 1984.

2. MC68000 16-Bit Microprocessor User's Manual, Third Edition, Prentice-Hall, Inc., 1982.

3. 68000 Relocatable Macro Assembler and Linking Loader Manual, Microtec, no date.

4. 68000 Simulator Manual, Microtec, no date.

APPENDIX A

ASCII CHARACTER SET

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	ç	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	¦
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

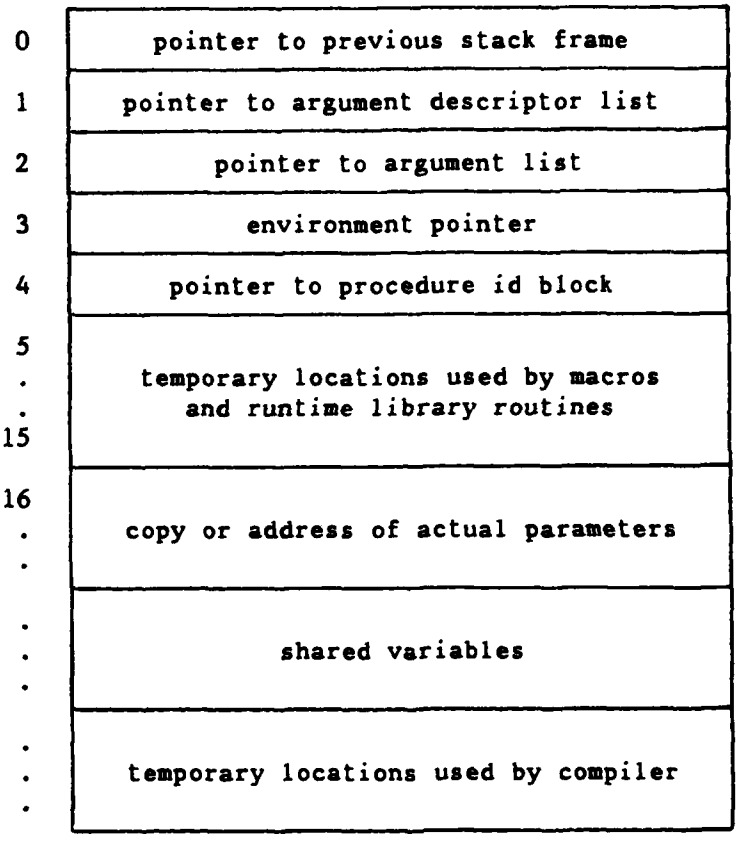
NUL	Null	DLE	Data Link Escape
SOH	Start of Heading	DC1	Device Control 1
STX	Start of Text	DC2	Device Control 2
ETX	End of Text	DC3	Device Control 3
EOT	End of Transmission	DC4	Device Control 4
ENQ	Enquiry	NAK	Negative Acknowledge
ACK	Acknowledge	SYN	Synchronous Idle
BEL	Bell	ETB	End of Transmission Block
BS	Backspace	CAN	Cancel
HT	Horizontal Tabulation	EM	End of Medium
LF	Line Feed	SUB	Substitute
VT	Vertical Tab	ESC	Escape
FF	Form Feed	FS	File Separator
CR	Carriage Return	GS	Group Separator
SO	Shift Out	RS	Record Separator
SI	Shift In	US	Unit Separator
SP	Space	DEL	Delete

The table above represents the ASCII character set. At the top of the table are hexadecimal digits (0 to 7), and to the left of the table are hexadecimal digits (0 to F). To determine the hexadecimal value of an ASCII

character, use the hexadecimal digit that corresponds to the row in the "units" position, and use the hexadecimal digit that corresponds to the column in the "16's" position. For example, the value of the character representing the equal sign is 3D.

Stack frames on the MC68000 are not adjacent. They are interwoven with the procedure linkage information on the processor stack that is supported by the MC68000.

The following diagram describes the structure of the stack frame. The offsets shown are in terms of THLL words.



B.3 TEMPORARIES

During the compilation process, it sometimes becomes necessary to employ temporary locations on the runtime stack (as shown in the diagram in Section B.2) for storage of intermediate results. Normally the temporaries follow the shared variables. However, if the amount of storage required for the whole stack frame exceeds 32768 bytes, the area for the temporaries is moved before the formal parameter area and the other areas are moved to higher displacements. This is done to ensure 'easy' access for temporaries. The addressing mode used for stack frame references allows a maximum displacement of 32767 bytes.

APPENDIX B

RUNTIME ENVIRONMENT ON THE MC68000

B.1 CONTROL SECTIONS

THLL programs are broken into three control sections on the MC68000. Control sections are used to identify characteristics associated with a data area or instruction area of a program. However, the control sections do not imply memory protections. The MC68000 hardware does not support such memory protection. The following table defines the MC68000 control sections.

Section	Purpose
O??	OWN data area
I??	Executable area
C??	Constant area

Each control section is relocatable. The MC68000 linker concatenates the same control section from all compile units.

Argument lists are not constant on the MC68000. Dynamic addresses are supplied to an argument list just before calling a procedure. The called procedure copies either the address of the argument (passed by reference) or the value of the argument (passed by value) to the runtime stack frame for that procedure invocation.

B.2 ALLOCATION OF STACK FRAMES

On entry to a procedure, a stack frame is allocated providing memory space for parameters, local variables, and temporaries. On exit from the procedure, the stack frame is released. General register A6 points to the stack frame for the current procedure invocation.

The first 16 THLL words on the stack frame are used to support the THLL procedure environment. This area also serves as a scratch area for THLL macros and library routines. The parameters to a procedure start at offset 16 on the stack frame. Shared variables are next on the stack frame, and temporary variables are normally last on each stack frame. The cross reference for a procedure indicates the hexadecimal byte offset of all user-defined items on the stack frame.

B.4 RUNTIME ERROR MESSAGES

The following error message is printed if any unsupported operations are used:

<operation> IS NOT SUPPORTED

The printing of this message occurs when using the Microtec MC68000 Simulator (Reference 4) and when using an MC68000 with the MACSBUG operating system. It is likely that this message will not be printed from an MC68000 within the TRIDENT II Fire Control System.

B.5 SPECIAL RUNTIME ERRORS

Some error conditions can be diagnosed only at runtime. MC TRICOMP generates the code to check for these errors. If an error occurs, the code contains an illegal instruction (opcode = X'4AFC'), followed by a 16-bit word containing a numeric value to identify the type of the error. The illegal instruction causes an illegal instruction exception to occur on the MC68000. The address saved on the linkage stack for the exception is the address of the illegal instruction. This arrangement lends itself to operating system analysis of the runtime errors. The numeric values and associated error conditions are shown in the following table.

Numeric Value	Error
1	Array subscript out of range
2	Stack subscript out of range
3	Stack overflow
4	Stack underflow

APPENDIX C

USING MC TRICOMP

C.1 INTRODUCTION

MC TRICOMP translates THLL code into MC68000 assembly code. This implies a three-step process in order to create an executable file. These steps are:

1. Compile THLL code producing MC68000 assembly code.

2. Assemble MC68000 code producing object code.

3. Link the object code into an executable file.

This appendix explains these steps. Additionally, it briefly describes how to produce the various reports available for THLL programs.

Note that filenames under VMS have the format Basename.EXT. Basename is the base filename for a set of related files that have different .EXT's extensions. Most VAX programs have a set of conventions governing the .EXT extensions. This is true for MC TRICOMP also. The user can override these conventions, but this is discouraged for configuration management reasons. The organization of a THLL program as presented in this appendix is not the only organization, but it is a logical one that should be considered seriously before deviating to another.

C.2 ORGANIZATION OF A THLL PROGRAM

A program written in THLL (targeted for the MC68000) consists of one or more compile units, all of which must be compiled, assembled, linked, and then executed. One usually thinks of a compile unit as a module, that is, a set of related procedures and data that can be compiled separately. On the VAX, a single compile unit maps nicely into a single EDT file having a .THL extension. Just as a compile unit maps into a file, a program (which is a collection of compile units) maps into a directory (which is a collection of files). Thus, on the VAX, a program written in THLL consists of one or more .THL files (each of which contains one compile unit) in a common directory. These .THL files are then compiled, assembled, linked, and then executed. Each step of this process creates additional files, most of which have the same base file names as the .THL files.

C.3 THE MC TRICOMP ENVIRONMENT

The MC TRICOMP environment is part of the VAX/VMS environment. It consists of a set of commands to produce an executable THLL program, a global cross reference report, and procedure call tree reports. The set of commands consists of the following:

1. TRICOMP - The command used to compile a THLL compile unit.

2. MCA - The command used to assemble the MC68000 assembly code corresponding to a THLL compile unit.

3. M68KLDR - The command used to link all the relocatable object files, generated by MCA, into one absolute executable file.

4. M68KSIM - The command used to invoke the MC68000 simulator to execute the absolute file produced by M68KLDR.

5. GXRUPDATE - The command used to combine .GXR files into a .MXD file.

6. GXRREPORT - The command used to produce a global cross reference report (.MXR file) from a .MXD file.

7. TREUPDATE - The command used to combine .TRE files into a .MTD file.

8. TREREPORT - The command used to produce a procedure call tree report (.MTR file) from a .MTD file.

9. NTREREPORT - The command used to produce a nested procedure call tree report (.NTR file) from a .MTD file.

These commands are available as all VMS commands are available. The THLL Reference Manual (Reference 1) and the VMS HELP command may be used to receive additional information about the above commands.

C.4 INPUT TO MC TRICOMP

C.4.1 Compile Units (.THL Files)

By default MC TRICOMP expects a compile unit in a file with a .THL extension. Each .THL file contains one THLL compile unit. A .THL file is an input source file that can be maintained with a VAX editor. It is recommended that each .THL file have a base filename corresponding to the compile unit name contained in the file.

C.4.2 MAIN Directive

The MAIN directive is used to specify which procedure is the program entry point. This directive must be placed in the compile unit containing the procedure which is being declared the MAIN procedure. There can be only one MAIN directive per program. The MAIN directive has the following format:

```
\ MAIN = procname                or
\\ MAIN = procname
```

C.4.3 Insert Files (.THI Files)

In addition to the .THL files that correspond to compile units, a THLL program often includes INSERT files. An insert file is an input source file that must have a .THI extension. It can be created simply by editing a file. In order to locate a .THI file associated with the insert declaration:

```
INSERT FILENAME(DIRNAME)
```

MC TRICOMP searches for DIRNAME in the following order:

1. The VMS logical name table is searched for a definition of DIRNAME. If it exists, that name is the name of the directory that contains the FILENAME.THI file. The DEFINE command in VMS is used to establish this relationship in the logical name table.

2. If DIRNAME is the default identifier, DEFAULT, the current default directory contains the FILENAME.THI file. DEFAULT could be defined in the logical name table in which case the check above would have associated DEFAULT to a (possibly different) directory.

3. Finally, it is assumed that DIRNAME is a subdirectory in the current default directory.

DIRNAME and FILENAME are truncated to 8 characters. This approach gives the program designer quite a bit of flexibility.

C.5 INVOCATION OF MC TRICOMP (.THL, .THI FILES)

After having created .THL and .THI files, MC TRICOMP can be invoked by the following command:

```
$ TRICOMP/MC Filename
```

where Filename is the base filename of a .THL file. It is recommended that the current default directory is set to be the directory containing the .THL file before invoking MC TRICOMP. Otherwise, the association is lost between

the .THL file and the files generated by MC TRICOMP.

C.6 OUTPUT FROM MC TRICOMP (.TLS, .MCS, .GXR, .TRE FILES)

The two file types, .THL and .THI, described above are created by the users as they develop their program. A .THL file contains a compile unit and is the input file to single invocation of MC TRICOMP. The .THI files contain information to be INSERTed into .THL files during a THLL compilation. MC TRICOMP compiles a single .THL file containing a single compile unit and produces four files. The default file extensions of the four files created by MC TRICOMP are:

.TLS - Compiled listing produced by MC TRICOMP

.MCS - MC68000 assembly code file corresponding to the THLL compile unit

.GXR - Global cross reference data for the THLL compile unit

.TRE - Procedure call tree data for the THLL compile unit

By default the base filename of each of these files is that of the .THL file containing the compile unit. MC TRICOMP creates the four output filenames by appending the above extensions to the base filename of the .THL file. See Section C.14 for ways to override the defaults.

The .TLS files produced by MC TRICOMP can be examined by the programmers to ensure that their development is proceeding as planned. The .MCS, .GXR, and .TRE files produced by MC TRICOMP are used as input to other programs.

C.7 INVOCATION OF MC68000 ASSEMBLER (.MCS, .OBJ, .LST FILES)

The .MCS files produced by MC TRICOMP contain the MC68000 assembly code (Reference 3) corresponding to the THLL compile units contained in the .THL files. These .MCS files must be assembled by the MC68000 assembler which produces an object code and a listing. To invoke this assembler to properly assemble a .MCS file created by MC TRICOMP, use the following command:

```
$ MCA Filename
```

where Filename is the base filename of a .MCS file created by MC TRICOMP. The file extensions of the two output files created by the MC68000 assembler are:

.LST - Assembled listing produced by the MC68000 assembler

.OBJ - Object code produced by the MC68000 assembler

Again, the base filenames of these two files are the same as the original .THL

file.

The .LST files are the assembler equivalent of the .TLS files. A THLL user ordinarily does not consult these .LST files, but they are valuable when searching for an extremely difficult bug.

C.8 INVOCATION OF MC68000 LOADER (.OBJ, .MAP, .ABS FILES)

The .OBJ files created by the assembler contain MC68000 object code corresponding to the THLL compile units contained in the .THL files. The .OBJ files are input to the MC68000 loader, which collects them into an executable file. Information on using the MC68000 loader is found in the 68000 Relocatable Macro Assembler and Linking Loader Manual (Reference 3). Input and output files for this loader are passed as parameters to the procedure. The following is the general command to invoke the MC68000 loader:

```
$ M68KLDR <input_file> [/ABS= /MAP= /COM=]
```

Input to the loader may be in the form of a command file or one or more relocatable object files. If input is via a command file (/COM=), then <input_file> must be specified as a null file (""). The /COM directive can not be used when an <input_file> is specified. As many qualifiers as desired may be specified and may appear in any order. The default base filename used if the qualifiers are not specified is L68K, with the corresponding extension(.CMD, .MAP, or .ABS).

It is recommended to use a command file because of the multiple sections generated by MC TRICOMP (I??, O??, C??). The following two steps can be used to execute the MC68000 loader.

1. Create a .CMD file. The following is an example of a command file, OUT.CMD :

```
SECT   I??,$1000
SECT   O??,$10000
SECT   C??,$20000
LOAD   DEMOMAIN.OBJ
LOAD   DEMOPRIME.OBJ
LOAD   MCSYSLIB:.
```

The first three commands define the starting absolute addresses of the various sections. The next two lines load the two modules DEMOMAIN.OBJ and DEMOPRIME.OBJ. The last command invokes the MC68000 system library MCSYSLIB.

2. Execute the loader:

```
$ M68KLDR "" /COM=OUT /ABS=OUT /MAP=OUT
```

Note: At least one blank must separate parameters and qualifiers, and no spaces may appear after

the slash (/) or around the equal sign (=).

This example selects the base filename of the executable file and the map file to be OUT. Thus, the output of this command is two files with the base filename OUT. The file extensions of these two files are:

.ABS - MC68000 absolute executable file

.MAP - Program load map

Note that up until this .ABS file, seven files have accumulated for each compile unit. For each Basename.THL file containing a compile unit, the THLL user derives Basename.TLS, Basename.MCS, Basename.GXR, Basename.TRE, Basename.LST, and Basename.OBJ. The ultimate goal is to combine a collection of .THL files into one executable file. Thus, even starting with three .THL files in a directory, only one .ABS file results (in the same directory).

C.9 RUNNING A PROGRAM (.ABS FILES)

There are two possibilities to choose from here. Either the absolute module can be executed on the VAX via the MC68000 simulator or it can be transported to the actual MC68000 for execution.

The command $ M68KSIM initiates the execution of the MC68000 simulator. Further information on this simulator can be obtained from the 68000 Simulator Manual (Reference 4).

C.10 PRODUCING A GLOBAL CROSS REFERENCE REPORT (.GXR, .MXD, .MXR FILES)

The global cross reference report presents an overview of the global symbols of a program. The actual individual compile unit line references for those symbols (as well as symbols local to a compile unit) can be found in the local cross reference at the end of the .TLS file. The global cross reference report is useful in determining which procedures and modules reference a particular global symbol. Symbols defined within an insert file are in a sense global; thus they are also included in the global cross reference report. The global cross reference report is developed from .GXR files in a two-step process.

The first step is to combine the .GXR files into a master cross reference data file (.MXD). This is done by invoking GXRUPDATE. The details of GXRUPDATE are presented in the THLL Reference Manual. The following command invokes GXRUPDATE:

$ GXRUPDATE Filename

where Filename is the base filename of the desired .MXD file. GXRUPDATE

combines the .GXR; files into the Filename.MXD file. The Filename.MXD file is the input file to the second step of this process.

The second step is to transform the .MXD file into a readable global cross reference report. This is done by invoking GXRREPORT. The details of GXRREPORT are presented in the THLL Reference Manual. The following command invokes GXRREPORT:

```
$ GXRREPORT Filename
```

where Filename is the base filename of the .MXD file. The file produced by GXRREPORT is Filename.MXR. This report can be examined and/or printed.

This two-step process allows for incremental updates to the global cross reference data. As the need arises to change selected compile units, the .THL files are changed, compiled using MC68000 TRICOMP, assembled, and linked. Later, the global cross reference data files are gathered by invoking GXRUPDATE to do a partial update to the .MXD file. GXRREPORT is used to form the new report. The changed compile units are reflected accordingly in the new report.

C.11 PRODUCING A PROCEDURE CALL TREE REPORT (.TRE, .MTD, .MTR FILES)

The procedure call tree report shows which procedures call which procedures. This report is produced in a two-step process that mirrors the two-step process for producing the global cross reference report.

The first step is to combine the .TRE files into a master procedure call tree data file (.MTD). This is done by invoking TREUPDATE. The details of TREUPDATE are presented in the THLL Reference Manual. The following command invokes TREUPDATE:

```
$ TREUPDATE Filename
```

where Filename is the base filename of the desired .MTD file. TREUPDATE combines the .TRE; files into the Filename.MTD file. The Filename.MTD file is the input file to the second step of this process.

The second step is to transform the .MTD file into a readable procedure call tree report. This is done by invoking TREREPORT. The details of TREREPORT are presented in the THLL Reference Manual. The following command invokes TREREPORT:

```
$ TREREPORT Filename
```

where Filename is the base filename of the .MTD file. The file produced by TREREPORT is Filename.MTR. This report can be examined and/or printed.

This two-step process allows for incremental updates to the procedure call tree data. As the need arises to change selected compile units, the .THL files are changed, compiled using MC68000 TRICOMP, assembled, and linked. Later, the procedure call tree data files are gathered by invoking TREUPDATE to do a partial update to the .MTD file. TREREPORT is used to form the new report. The changed compile units are reflected accordingly in the new report.

C.12 PRODUCING A NESTED PROCEDURE CALL TREE REPORT (.TRE, .MTD, .NTR FILES)

The nested procedure call tree report shows which procedures call which procedures in a nested format. This report is produced in a two-step process that mirrors the two-step process for producing the regular procedure call tree report.

The first step is exactly the same as the first step of producing a procedure call tree report. The first step is to combine the .TRE files into a master procedure call tree data file (.MTD). This is done by invoking TREUPDATE as shown above. If the .MTD file has been created for a regular procedure call tree report, it can be used for a nested procedure call tree report.

The second step is to transform the .MTD file into a readable nested procedure call tree report. This is done by invoking NTREREPORT. The details of NTREREPORT are presented in the THLL Reference Manual. The following command invokes NTREREPORT:

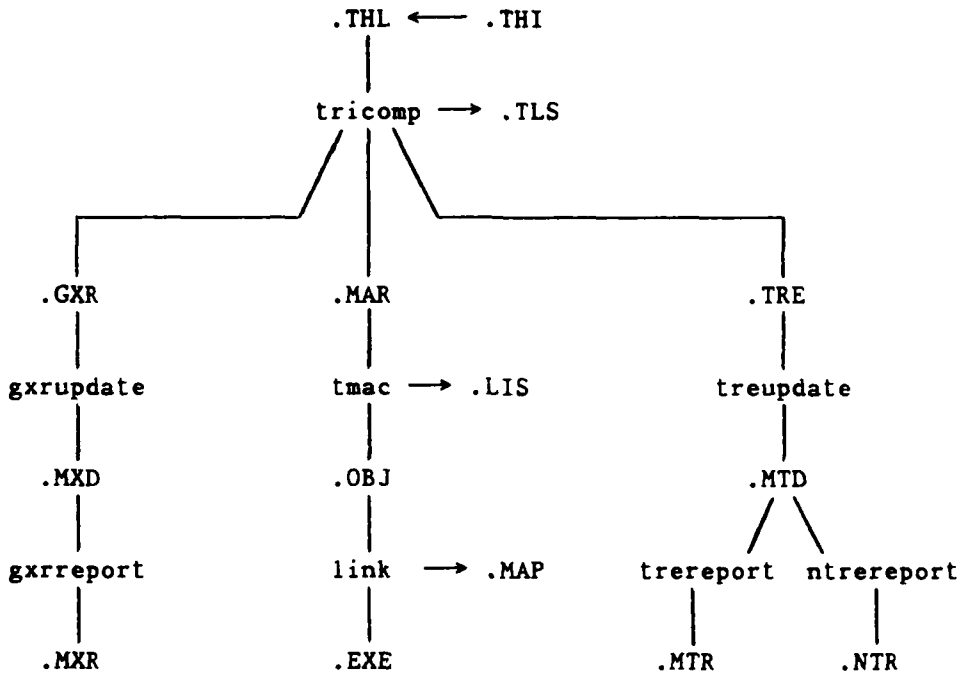
```
$ NTREREPORT Filename
```

where Filename is the base filename of the .MTD file. The file produced by NTREREPORT is Filename.NTR. This report can be examined and/or printed.

As in producing a regular procedure call tree report, this two-step process allows for incremental updates to the nested procedure call tree data.

C.13 DIAGRAM OF A THLL PROGRAM'S FILE RELATIONSHIPS

The preceding description is summarized pictorially in the following diagram. Flow is downward.



C.14 ADVANCED INVOCATIONS OF TRICOMP (OVERRIDING DEFAULTS)

The preceding sections of this appendix describe how to use MC TRICOMP, accepting the defaults. Sometimes defaults are not sufficient and sometimes defaults constrict a programmer's creativity. If the need arises, MC TRICOMP provides ways to override the defaults. All command qualifiers can be seen in the THLL Reference Manual (Reference 1) or via the VMS HELP TRICOMP command.

C.14.1 Suppressing TRICOMP Generated Files

Often utility programs consist of a small number (sometimes one) of compile units. In these cases, the .GXR and .TRE files are cumbersome (or not necessary). The following command does not produce a .GXR and a .TRE file.

```
S TRICOMP/MC/NOGXR/NOTRE UTIL
```

C.14.2 Parallel Directory Organization

Sometimes a program is so large that keeping all of the files in the same directory would be unmanageable. A valid approach is to view a program as a collection of parallel directories where:

1. [PROGRAM.THL] contains the .THL files,
2. [PROGRAM.TLS] contains the .TLS files,
3. [PROGRAM.MCS] contains the .MCS files,
4. [PROGRAM.GXR] contains the .GXR files, and
5. [PROGRAM.TRE] contains the .TRE files.

The following command sprays the .TLS, .MCS, .GXR, and .TRE files into the appropriate directories for [PROGRAM.THL]FILE1.THL.

```
$ TRICOMP/MC/TLS=[PROGRAM.TLS]/MCS=[PROGRAM.MCS]-
S_/GXR=[PROGRAM.GXR]/TRE=[PROGRAM.TRE]  [PROGRAM.THL]FILE1
```

C.14.3 Overriding Default File Extensions

Occasionally programmers believe that their own conventions are more expressive than the default conventions. The following command compiles a THLL compile unit contained in file PROG.SRC and places the MC68000 assembly code in file PROG.ASM.

```
$ TRICOMP/MC/NOTRE/NOGXR/MCS=.ASM  PROG.SRC
```

Note that the THLL listing is placed in the default PROG.TLS file.

C.14.4 Example Compile and Assemble Command Procedure

MC TRICOMP produces MC68000 assembly code which must be assembled by the MC68000 assembler. Many programmers are not interested in the assembly code as they just want the .OBJ files which can be linked to form the .ABS file. Thus many programmers consider this two-step process of compiling and assembling as one, where MC TRICOMP produces a .OBJ file from a .THL file. If there are no compile errors in the .THL file, the following command procedure produces a .OBJ file from a .THL file.

```
$ ON WARNING THEN EXIT
$ TRICOMP/MC/NOGXR/NOTRE 'P1'
$ MNAME = F$PARSE(P1,,,"NAME")
$ MCA 'MNAME'
```

```
$ DEL 'MNAME'.LST;
$ DEL 'MNAME'.MCS;
$ EXIT
```

C.14.5 $SEVERITY Returned from MC TRICOMP

The preceding example command file relies upon the $SEVERITY system symbol. When TRICOMP exits to VMS, $SEVERITY is set to indicate how the compilation went. The following values can be found in $SEVERITY.

1. $SEVERITY = 1 implies a normal compilation
2. $SEVERITY = 0 implies normal compilation with compile errors
3. $SEVERITY = 2 implies abnormal compilation
4. $SEVERITY = 4 implies one of the pertinent files could not be found or opened

Note that if the \\ ABORT directive is contained within a compile unit, then a $SEVERITY of 0 becomes a 2 and a $SEVERITY of 2 becomes a 4.

C.15 EXAMPLE SESSION ON USING MC TRICOMP

The following is an example session using the MC TRICOMP environment. All of the features discussed above are covered in the example. Users confused by the above descriptions may want to create the necessary files and actually perform this session. The necessary files are DEMOMAIN.THL, DEMOPRIME.THL, and EXTEND.THI. These files are listed in this appendix following this example. An understanding of VAX facilities for creating files is assumed. A slight twist on the example is to try putting the EXTEND.THI file in the same (or parallel) directory as the .THL files.

```
$ SD.

  UDISK1:[username]

$ CREATE/DIR [.DEMO]
$ SD.DEMO

  UDISK1:[username.DEMO]

$
$ ! create DEMOMAIN.THL and DEMOPRIME.THL (listed in appendix)
$ ! using one of the VAX editors
$
$ DIR
```

```
Directory UDISK1:[username.DEMO]

DEMOMAIN.THL;1   DEMOPRIME.THL;1

Total of 2 files.
$ CREATE/DIR [.INSERTS]
$ SD.INSERTS

  UDISK1:[username.DEMO.INSERTS]

$
$ ! create EXTEND.THI (listed in appendix) using one of the
$ ! VAX editors
$
$ SD. ! sets home directory as default directory

  UDISK1:[username]

$ SD [.DEMO]

  UDISK1:[username.DEMO]

$
$ ! show the program's structure
$
$ DIR

Directory UDISK1:[username.DEMO]

DEMOMAIN.THL;1   DEMOPRIME.THL;1  INSERTS.THI;1

Total of 3 files.
$ SD.INSERTS ! subdirectory containing insert files

  UDISK1:[username.DEMO.INSERTS]

$ DIR

Directory UDISK1:[username.DEMO.INSERTS]

EXTEND.THI;1

Total of 1 file.
$ SD ^

  UDISK1:[username.DEMO]

$
$ ! compile .THL files (each has one compile unit)
$
$ TRICOMP/MC DEMOMAIN
 BEGIN MC TRICOMP
```

```
 NORMAL COMPILATION  DEMOMAIN
 END MC TRICOMP
$ TRICOMP/MC DEMOPRIME
 BEGIN MC TRICOMP
 NORMAL COMPILATION  DEMOPRIME
 END MC TRICOMP
$ DIR

Directory UDISK1:[username.DEMO]

DEMOMAIN.GXR;1     DEMOMAIN.MCS;1     DEMOMAIN.THL;1     DEMOMAIN.TLS;1
DEMOMAIN.TRE;1     DEMOPRIME.GXR;1    DEMOPRIME.MCS;1    DEMOPRIME.THL;1
DEMOPRIME.TLS;1    DEMOPRIME.TRE;1    INSERTS.DIR;1

Total of 11 files.
$
$ ! assemble the .MCS files generated by MC TRICOMP
$
$ MCA DEMOMAIN
  ASSEMBLER ERRORS =     0
$ MCA DEMOPRIME
  ASSEMBLER ERRORS =     0
$ DIR

Directory UDISK1:[username.DEMO]

DEMOMAIN.GXR;1     DEMOMAIN.LST;1     DEMOMAIN.MCS;1     DEMOMAIN.OBJ;1
DEMOMAIN.THL;1     DEMOMAIN.TLS;1     DEMOMAIN.TRE;1     DEMOPRIME.GXR;1
DEMOPRIME.LST;1    DEMOPRIME.MCS;1    DEMOPRIME.OBJ;1    DEMOPRIME.THL;1
DEMOPRIME.TLS;1    DEMOPRIME.TRE;1    INSERTS.DIR;1

Total of 15 files.
$
$ ! link the .OBJ files into a .ABS file
$
$ EDT DEMO.CMD
Input file does not exist
[EOB]
SECT  I??,$2000
SECT  O??,$3500
SECT  C??,$5000
LOAD  DEMOMAIN.OBJ
LOAD  DEMOPRIME.OBJ
LOAD  MCSYSLIB:.
^Z        ! <CONTROL Z>
*EX

$ M68KLDR "" /COM=DEMO /ABS=DEMO /MAP=DEMO


  **LOAD COMPLETE
```

```
$ DIR

Directory UDISK1:[username.DEMO]

DEMO.ABS;1          DEMO.MAP            DEMOMAIN.GXR;1      DEMOMAIN.LST;1
DEMOMAIN.MCS;1      DEMOMAIN.OBJ;1      DEMOMAIN.THL;1      DEMOMAIN.TLS;1
DEMOMAIN.TRE;1      DEMOPRIME.GXR;1     DEMOPRIME.LST;1     DEMOPRIME.MCS;1
DEMOPRIME.OBJ;1     DEMOPRIME.THL;1     DEMOPRIME.TLS;1     DEMOPRIME.TRE;1
INSERTS.DIR;1

Total of 17 files.
$
$ ! run the DEMO program on the MC68000 simulator
$
$ M68KSIM
 68000 INTERACTIVE SIMULATOR VER 3.0

 RECOVER EXISTING MEMORY FILE? ANSWER Y OR N:
-N
-MAX 3FF10H
-L DEMO.ABS
-LI 30000H
-OFAS : 3FF03H
-S SP 3FF00H
-E
00000002
*00000003
*00000005
*00000007
*0000000B
*0000000D
*00000011
*00000013
*00000017
*0000001D
*0000001F
*00000025
*00000029
*0000002B
*0000002F
*00000035
*0000003B
*0000003D
*
 *** ADDRESS OUT OF RANGE
       ! <instruction data dump>
-EX
$
$ ! Produce a Global Cross Reference Report
$
$ GXRUPDATE DEMO
$ DEL *.GXR;       ! Not needed after the update
```

```
$ DIR

Directory UDISK1:[username.DEMO]

DEMO.ABS;1          DEMO.MAP;1          DEMO.MXD;1          DEMOMAIN.LST;1
DEMOMAIN.MCS;1      DEMOMAIN.OBJ;1      DEMOMAIN.THL;1      DEMOMAIN.TLS;1
DEMOMAIN.TRE;1      DEMOPRIME.LST;1     DEMOPRIME.MCS;1     DEMOPRIME.OBJ;1
DEMOPRIME.THL;1     DEMOPRIME.TLS;1     DEMOPRIME.TRE;1     INSERTS.DIR;1

Total of 16 files.
$ GXRREPORT/TITLE="Demonstration Use of MC TRICOMP" DEMO
$ DIR

Directory UDISK1:[username.DEMO]

DEMO.ABS;1          DEMO.MAP;1          DEMO.MXD;1          DEMO.MXR;1
DEMOMAIN.LST;1      DEMOMAIN.MCS;1      DEMOMAIN.OBJ;1      DEMOMAIN.THL;1
DEMOMAIN.TLS;1      DEMOMAIN.TRE;1      DEMOPRIME.LST;1     DEMOPRIME.MCS;1
DEMOPRIME.OBJ;1     DEMOPRIME.THL;1     DEMOPRIME.TLS;1     DEMOPRIME.TRE;1
INSERTS.DIR;1

Total of 17 files.
$
$ ! Produce a Procedure Call Tree Report
$
$ TREUPDATE DEMO
$ DEL *.TRE;        ! Not needed after the update
$ DIR

Directory UDISK1:[username.DEMO]

DEMO.ABS;1          DEMO.MAP;1          DEMO.MTD;1          DEMO.MXD;1
DEMO.MXR;           DEMOMAIN.LST;1      DEMOMAIN.MCS;1      DEMOMAIN.OBJ;1
DEMOMAIN.THL;1      DEMOMAIN.TLS;1      DEMOPRIME.LST;1     DEMOPRIME.MCS;1
DEMOPRIME.OBJ;1     DEMOPRIME.THL;1     DEMOPRIME.TLS;1     INSERTS.DIR;1

Total of 16 files.
$ TREREPORT/TITLE="Demonstration Use of MC TRICOMP" DEMO
$ DIR

Directory UDISK1:[username.DEMO]

DEMO.ABS;1          DEMO.MAP;1          DEMO.MTD;1          DEMO.MTR;1
DEMO.MXD;           DEMO.MXR;           DEMOMAIN.LST;1      DEMOMAIN.MCS;1
DEMOMAIN.OBJ;1      DEMOMAIN.THL;1      DEMOMAIN.TLS;1      DEMOPRIME.LST;1
DEMOPRIME.MCS;1     DEMOPRIME.OBJ;1     DEMOPRIME.THL;1     DEMOPRIME.TLS;1
INSERTS.DIR;1

Total of 17 files.
$
$ ! Produce a Nested Procedure Call Tree Report
$ ! Note the .MTD file already exists
```

```
$
$ NTREREPORT/TITLE="Demonstration Use of MC TRICOMP" DEMO
$ DIR

Directory UDISK1:[username.DEMO]

DEMO.ABS;1          DEMO.MAP;1          DEMO.MTD;1          DEMO.MTR;1
DEMO.MXD;           DEMO.MXR;           DEMO.NTR;1          DEMOMAIN.LST;1
DEMOMAIN.MCS;1      DEMOMAIN.OBJ;1      DEMOMAIN.THL;1      DEMOMAIN.TLS;1
DEMOPRIME.LST;1     DEMOPRIME.MCS;1     DEMOPRIME.OBJ;1     DEMOPRIME.THL;1
DEMOPRIME.TLS;1     INSERTS.DIR;1

Total of 18 files.
```

C.15.1 Example THLL Compile Unit (DEMOMAIN.THL FILE)

```
DEMOMAIN
     BEGIN

     \\ MAIN = MAIN

     COMMENT THIS PROGRAM DETERMINES THE PRIME NUMBERS FROM 2 TO 63.
             THE METHOD USED IS A FORM OF SIEVE ERATOSTHENES.  THE
             NUMBERS TO SEARCH ARE IN THE ARRAY NUM.  EACH BIT IN
             THE ARRAY CAND CORRESPONDS TO AN ENTRY IN NUM.  AS LONG
             AS THE CAND BIT IS SET, THE NUM ENTRY IS A CANDIDATE
             FOR A PRIME NUMBER.  IF AN ENTRY IN NUM IS FOUND TO BE
             A COMPOSITE NUMBER, ITS BIT IN CAND IS CLEARED.  ;

     INSERT EXTEND(INSERTS) ;

     GLOBAL CAND,NUM ;

     OWN INTEGER ARRAY CAND(1),NUM(63) ;

     EXTERNAL PROCEDURE PRIME ;
     EXTERNAL PROCEDURE PRTINT(INTEGER) ;

     INTEGER COMPONENT PRIMECAND (OFFSET 0,FIELD(0,1)) ;

     GLOBAL MAIN ;

     DEFINE PROCEDURE MAIN ;
          BEGIN
          INTEGER I ;
          POINTER P ;
          FOR I=0 STEP 1 UNTIL 63 DO
               NUM(I) = I+2
          ENDDO ;
          FOR I=0 STEP 1 UNTIL 1 DO
               CAND(I) = X'FFFFFFFF'
          ENDDO ;
          PRIME ;
          P = LOC CAND(0) ;
          FOR I=0 STEP 1 UNTIL 63 DO
               IF PRIMECAND(P,I) EQL 1 THEN
                    PRTINT(NUM(I))
               IFEND
          ENDDO ;
          END ;
     END FINIS
```

C.15.2 Example THLL Compile Unit (DEMOPRIME.THL FILE)

```
DEMOPRIME
     BEGIN
     COMMENT  THIS MODULE CONTAINS THE PROCEDURE FOR CALCULATING THE
              PRIME NUMBERS, ALONG WITH THE MODULE USED FOR PRINTING
              THE RESULTS. ;

     INSERT EXTEND(INSERTS) ;

     EXTERNAL PROCEDURE TERM(ALPHA) ;

     EXTERNAL INTEGER ARRAY CAND(1) ;
     EXTERNAL INTEGER ARRAY NUM(63) ;

     ALPHA COMPONENT A0(OFFSET 1);
     INTEGER COMPONENT P0(FIELD(0,4),OFFSET 0);
     INTEGER COMPONENT PRIMECAND(OFFSET 0,FIELD(0,1)) ;

     GLOBAL PRIME ;

     DEFINE PROCEDURE PRIME ;
          BEGIN
          INTEGER CURPRIME,J,STOPVAL ;
          POINTER P ;
          P = LOC CAND(0) ;
          STOPVAL = 8 ;          /* SQRT(64)=8 */
          CURPRIME = 0 ;
          WHILE NUM(CURPRIME) LEQ STOPVAL DO
               BEGIN
               /* REMOVE MULTIPLES OF CURRENT PRIME FROM CANDIDATES */
               FOR J = CURPRIME+1 STEP 1 UNTIL 63 DO
                    IF PRIMECAND(P,J) EQL 1 THEN
                         IF (NUM(J) MOD NUM(CURPRIME)) EQL 0 THEN
                              PRIMECAND(P,J) = 0
                         IFEND
                    IFEND
               ENDDO ;
               /* FIND NEXT PRIME TO PROCESS */
               CURPRIME = CURPRIME + 1 ;
               END
          ENDDO ;
          END ;

     GLOBAL PRTINT;

     DEFINE PROCEDURE PRTINT(INT) ;
          VALUE INT;
          INTEGER INT;

          BEGIN
          OWN ALPHA PRTBUF(7) ;
```

```
     OWN POINTER PTR,PTR1 ;
     OWN INTEGER I ;
     OWN INTEGER TEMP ;
     PRESET PRTBUF=#'12345678' ;
     PTR = LOC INT ;
     PTR1 = LOC PRTBUF ;
     FOR I = 0 STEP 1 UNTIL 7 DO
          BEGIN
          TEMP = P0(PTR,I) ;
          IF TEMP GRT 9 THEN
               A0(PTR1,I) = TEMP + 55
          ELSE
               A0(PTR1,I) = TEMP + 48
          IFEND;
          END
     ENDDO ;
     TERM(PRTBUF);
     END;
END FINIS
```

C.15.3 Example THLL Insert File (EXTEND.THI FILE)

```
\\ BOUNDS = 0 , XREF = 3

COMMENT THESE SYNONYM DECLARATIONS ARE USED TO EXTEND THLL ;

SYNONYM
     BEGIN
     NEXTCASE = / , / ;
     ELSEIF = / , / ;
     BOOLEAN = / INTEGER / ;
     ENDDO = / / ;
     END ;
```

C.15.4 Example Global Cross Reference Report (DEMO.MXR FILE)

GLOBAL CROSS REFERENCE Demonstration Use of MC TRICOMP PAGE 1

SYMBOL NAME	ATTRIBUTES	DEFINED IN DECK		USED IN DECK	USED IN PROCEDURE	REFERENCES USED	CHNG
CAND	IA(1)	DEMOMAIN		DEMOMAIN	MAIN	1	1
				DEMOPRIME	PRIME	1	
ENDDO	SYNONYM	EXTEND	I	DEMOMAIN	MAIN	3	
				DEMOPRIME	PRIME	2	
					PRTINT	1	
GEM	SYNONYM	EXTEND	I	DEMOMAIN		1	
				DEMOPRIME		1	
MAIN	P(0)	DEMOMAIN					
NUM	IA(63)	DEMOMAIN		DEMOMAIN	MAIN	1	1
				DEMOPRIME	PRIME	3	
PRIME	P(0)	DEMOPRIME					
				DEMOMAIN	MAIN	1	
PRTINT	P(1)	DEMOPRIME					
				DEMOMAIN	MAIN	1	
TERM	X			DEMOPRIME	PRTINT	1	

C.15.5 Example Procedure Call Tree Report (DEMO.MTR FILE)

```
PROCEDURE CALL TREE Demonstration Use of MC TRICOMP      PAGE   1

      MAIN       G          DEMOMAIN
                 PRIME      *
                 PRTINT

      PRIME      G          DEMOPRIME

      PRTINT     G          DEMOPRIME
                 TERM       X
```

C.15.6 Example Nested Procedure Call Tree Report (DEMO.NTR FILE)

```
PROCEDURE CALL TREE Demonstration Use of MC TRICOMP       PAGE    1
MAIN TREE                                        10-JUL-1984 10:18:01


LINE  LEVEL ROUTINE
----  ----- -------

   1      1 MAIN
   2      2    PRIME
   3      3       PRTINT
   4      4        -TERM
   5      2    PRTINT      (3)
```

END

FILMED

6-85

DTIC